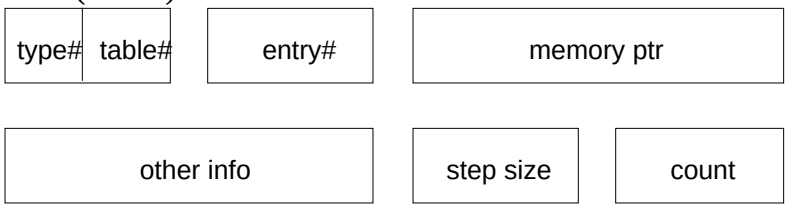
Data Access Table Formats

Data pool preparation

Thu, Jul 25, 1996

Introduction

The Data Access Table is used to specify what happens within a station every cycle to prepare the data pool. Included in this is a mechanism for executing all enabled closed loops and server code. At the start of a periodic cycle, the Update Task is executed. As part of its work, entries in the DAT are processed from beginning to end. Each entry is an "instruction" to be interpreted before moving on to the next entry / instruction. The generic pattern of such entries is as follows, shown as 8 (16-bit) words:

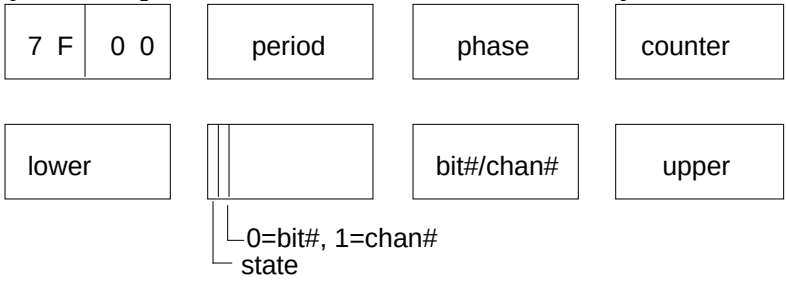


The hi byte of the first word specifies the entry type#. The lo byte of the first word is a system table# if it is in the range 00–1F. The meanings of all other fields depend upon the entry type#. The above layout of these entries is only an example. Normal entry type#s are positive integers in the range 01–7F. Entry type#s of 00 or in the range 80–FF are invalid and are ignored during DAT processing. Entry type 7F is the condition entry.

The table# is normally either 00 (for the channel entries in the Analog Data Table) or 05 (for the byte entries in the Binary Data Table). The entry# is the entry of the first data word targeted. The data words are copied into successive entries of the destination table when a count is specified. Most often, the entry# is a channel#. Some types use an address pointer which provides a hardware board address or other memory pointer. The 7th word is often a step size when a memory ptr is used. The count word is usually a loop count of the number of consecutive channel readings (destination table entries) to be filled.

Condition entry

Entry type 7F is a special entry that is used to enable / disable an internal flag that determines whether a non-7F entry is processed or skipped. At the beginning of DAT processing, this internal flag is initialized to disabled, so that a 7F entry must occur before any non-7F entries will be interpreted for processing. The only result of processing a 7F entry, which by definition is never disabled, is to enable or disable the internal flag that determines whether a subsequent non-7F entry will be processed. The format of a 7F entry is as follows:



At its simplest, this entry merely specifies the period word in cycles. For

processing every cycle, use 1. Such is typically the first entry in a DAT. If processing is desired every other cycle, use period=2. The other options allow for enabling the internal flag based upon the state of a bit, or the value of an analog channel being inside or outside a specified range. For more information, see the document RDATA Periodicity.

Overview

As an overview of the entry types available for use as DAT instructions, here is a list by type#:

01	Multiplexed A/D used by Linac
02	(n.u.)
03	Read memory words by bytes
04	Read binary bytes via address list
05	Shift data words
06	Adjust nonlinear RF diode readings
07	Zero-data pedestal adjustment
08	Compute ratio
09	Compute product
0A	Compute sum
0B	Compute difference
0C	Process 1553 command list
0D	Auto-setting from memory
0E	Wait, post-process 1553 data
0F	Average sequence of readings
10	(n.u.)
11	Analog Devices A/D board
12	Sample datapool
13	Read memory words by words
14	High Voltage Digitizer (obs)
15	Beam status counter
16	Capture data on selected cycles (obs)
17	Timer channel clock event counts (obs)
18	Timer channel clock events (obs)
19	AMD9513 timer delays (obs)
1A	Read single bytes of memory
1B	Read words—mask,shift,BCD options
1C	Read clock events from clock board (obs)
1D	Invoke local applications
1E	Insert data into memory words
1F	De-multiplex data words
20	Send data request to SRMs
21	Wait for SRM data reply
22	Map SRM data into data pool
23	(n.u.)
24	Compute counter differences
25	Copy setting word
26	Assemble combined status words
27	Copy memory into data stream
28	Copy from IRM A/D circular buffer
29	De-multiplex binary data bytes
2A	Copy words memory-memory
2B	Copy bytes memory-memory
2C	Copy FIFO to memory
2D	Save all readings in present cycle

Some entry types access data from various hardware interfaces. Some modify data already collected in various ways. In particular, entry 1D allows for processing all local applications, some of which may generate output data for inclusion in the data pool. Much of the particularization, or configuration, of a station stems from the design of its data access table entries.

Editing DAT entries

Armed with the detailed specifications, one can enter DAT entries using a memory dump page. If this is done, one should take care, as the DAT is scanned every cycle, so it is "live." As changes made in this way are usually made one word at a time, it may be wise to disable an entry while it is being modified, say by setting the $80 bit in the type# byte. Remove this sign bit when the rest of the entry is ready. This form of raw editing of the DAT is not for the squeamish.

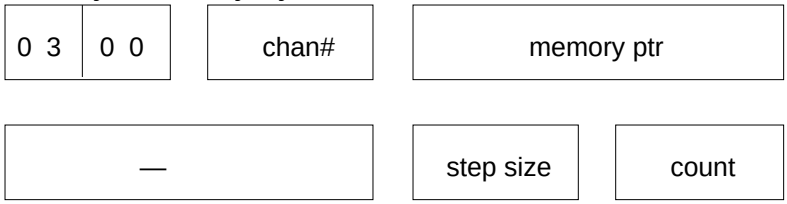
Another means of editing the DAT is provided by a Unix tool called xxxxx. It operates by reading up the entire DAT and producing a text file version of it, which is then edited and downloaded all at once. See document xxxxxx.

DAT entry formats

A brief description follows for each DAT entry type, grouped into related types. In some cases, additional details can be found in other related documents.

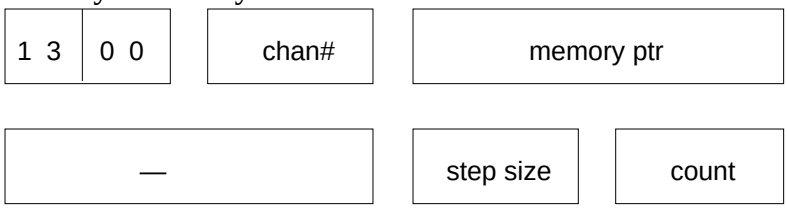
Accessing memory data

Read memory words by bytes



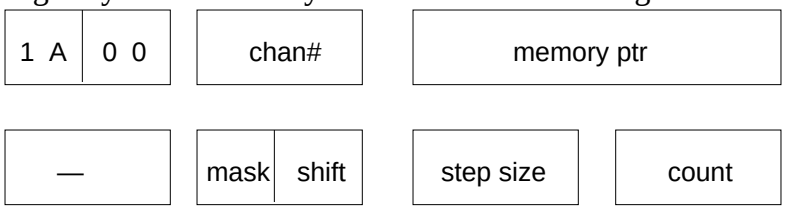
Words are copied (accessed by bytes) starting at the memory address given, advancing by the step size for each destination table entry (*chan#*). If the *memory ptr* refers to data stored in consecutive words, the *step size* would be 2.

Read memory words by words



Words are copied (accessed by words) starting at the memory address given, advancing by the step size for each destination table entry (*chan#*). If the *memory ptr* refers to data stored in consecutive words, the *step size* would be 2.

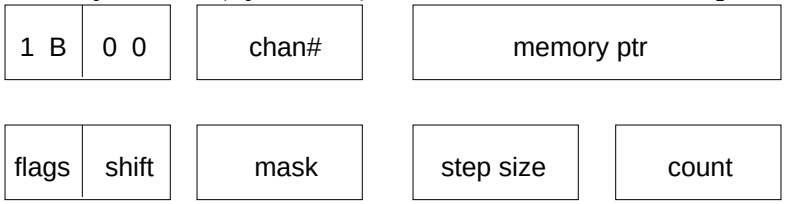
Read single bytes of memory and convert to reading words.



For the range of selected channels, read a byte from memory, apply an optional mask, shift an optional amount and use the resulting 16-bit word as a reading. If the *mask* is zero, no masking will be applied. If the *shift* count is negative, a right shift of (*–shift*) bits is indicated, starting from the data byte positioned in the hi

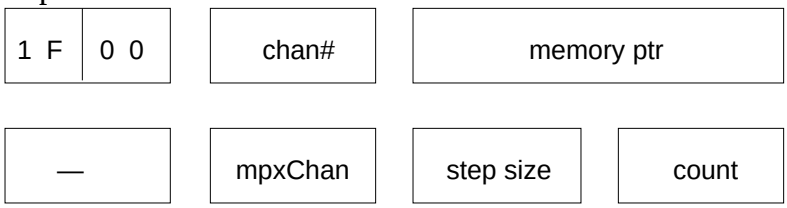
byte of the word and zero in the lo byte. If the *shift* count is positive, a left shift of that many bits is indicated, starting from the byte positioned in the lo byte of the word and zero in the hi byte. The *step size* is used to advance the *memory ptr* when more than one byte is accessed (*count* > 1).

Read memory words (by words) with mask, shift, BCD options



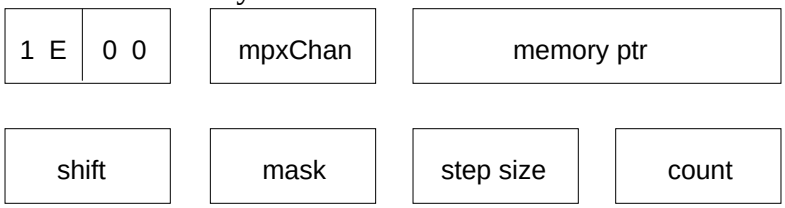
Copy words of memory (accessed by 16-bit read cycles) into consecutive channel readings. Apply optional *mask* (0 treated as $FFFF), optional *shift* (positive=left, negative=right, zero=none), and optional BCD-to-binary conversion (*flags*=$80 to enable conversion). The *step size* advances *memory ptr* for *count* words.

De-multiplex data words



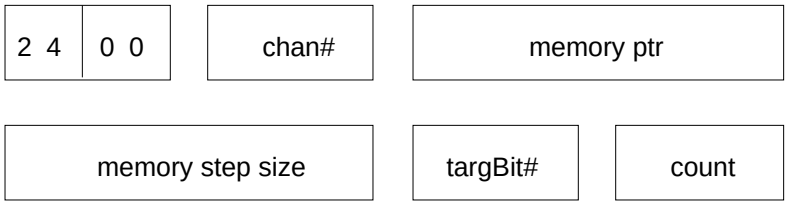
De-multiplex words of memory data according to the value of the *mpxChan*. The value of *mpxChan*, for example, may range from 0–F on successive cycles. Data from memory (count words using step size) is copied into the readings of channels numbered from (*chan#*mpxValue*) to (*chan#*mpxValue* + *count* – 1). This is useful when the hardware interface furnishes multiplexed data according to a value supplied on digital control lines. Type 1E may be used to place the proper value on the control lines.

Insert data into memory words



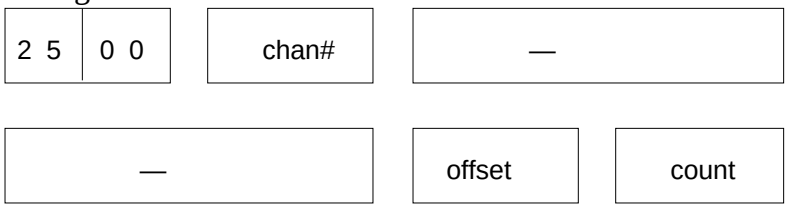
Sample masked value from *mpxChan* and insert into memory words. The reading of *mpxChan* is masked by *mask* and left-shifted by *shift* bits (rotated as a 16-bit word, so use 16–n to shift right) and inserted into the target memory word(s). The bits outside the mask in the target word(s) are not modified.

Compute counter differences



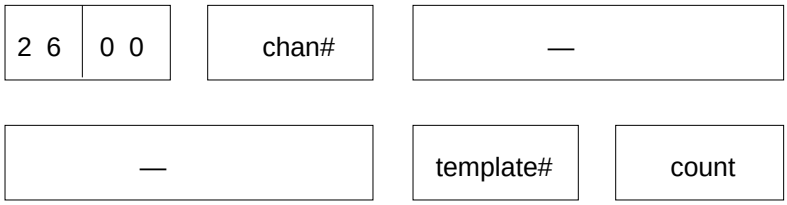
Monitors memory word counter differences. This can be used to monitor whether an associated cpu in the same VME crate is still working by watching a counter word at *memory ptr* that the cpu increments regularly. Optionally, if *targBit#* is nonzero, a bit# can be set if the difference from last time is nonzero, and cleared if the difference is zero. When *count* > 1, additional memory counter addresses are obtained from using memory *step size*, and successive bit#s are used when *targBit#* is nonzero. The memory of what the word read last time is retained in the setting word of the associated target difference *chan#*, so such channels cannot be settable.

Copy setting word



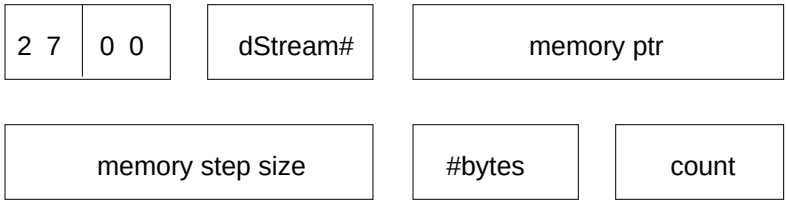
Copy setting (or other) field values into reading fields of successive analog channels. The *offset* value is the offset to the required field in the ADATA table entry relative to the reading field. Use *offset* = 2 for setting field values.

Assemble combined status words

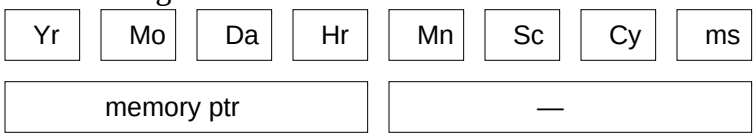


Assemble words of status from collections of bits found in the BBYTE table, using templates found in the CSTAT table #24. Each status word is built from a template found in this table. The reading of *chan#* is built from *template#*, and the process is repeated for successive channels and templates according to *count*. The template is an entry from the CSTAT table, each of which consists of up to 8 specifications of 4 bytes each. Each specification is a Byte# word, followed by a shift count byte and a mask byte. See document Composite Digital Status for more details.

Copy memory blocks into data stream



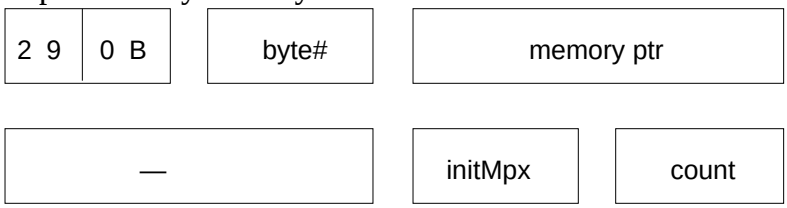
Copy a block of memory of size *#bytes* from *memory ptr* into a data stream with index *dStream#*. The beginning of the record written contains a 16-byte header with the following format:



This header includes the time-of-day the record was written, followed by the memory address from which the block was copied. If *count* > 1, then multiple blocks of memory can be so captured, each including a header. In this case, subsequent block source memory addresses are derived using the memory *step size*. The data stream should be defined in the DSTRM table to have records whose size reflects both the header size and the data block size. For example, if 1024-byte blocks of memory were to be captured into a data stream, the data stream record size as defined would be $410. The time-of-day format is BCD, except for the *ms* byte that holds the residual milliseconds of the present cycle. The *Cy* byte ranges from $00–14, indicating the present 15Hz cycle.

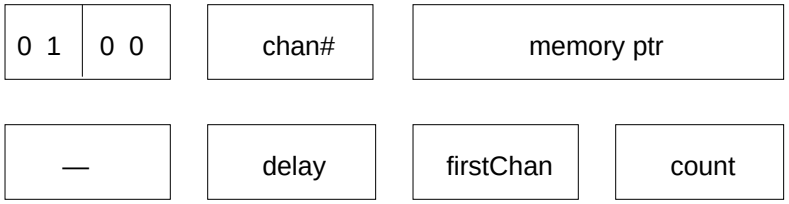
Access to specific hardware interfaces

De-multiplex binary data bytes



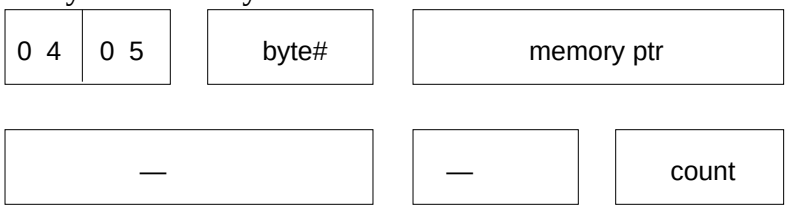
This entry assumes a simple hardware interface for multiplexing binary data bytes. The *memory ptr* is the address of a multiplexed data byte; *memory ptr+1* is the address of the multiplex select byte. The *initMpx* is the initial value of the multiplex select byte; subsequent values are merely incremented from that. The *byte#* is the initial entry used in the BADDR table for obtaining the target addresses for the data byte read from the multiplexed data byte. This simple multiplexing scheme may be used to bring in many digital data bytes using only two bytes of I/O interface. (Each IRM, for example, includes an interface to eight bytes of digital I/O. In the Fermilab Booster HLRF system, two of these bytes are used in this scheme to bring in 16 bytes of multiplexed digital status.)

Multiplexed A/D used in Fermilab Linac



This entry accesses A/D data as interfaced via the original multiplexed A/D system in use at the Fermilab Linac. The SRMs have since been used to read this data, so this entry is no longer needed.

Read binary raw data bytes

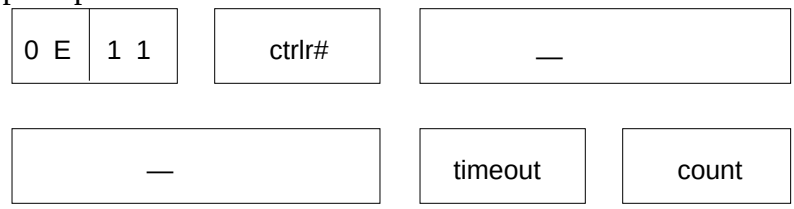


An array of byte addresses (usually the BADDR table) specified by memory ptr contains pointers to consecutive bytes of binary status data. They are treated as memory-mapped data bytes unless the high byte of the address found is $80, 81, or 82, which carry special significance. [If the high byte is 80, the entry is assumed to be a pointer to a 1553 data byte in a 1553 command block on the 1553 controller board's memory. If the high byte is 81, the next byte is an SRM address, and the last two bytes is the control value needed to be sent to the SRM for setting the byte. If the high byte is 82, the next three bytes specify parameters needed for PLCQ message queue processing.] In other cases, the 4-byte entry is a memory address that is accessed to obtain the data byte stored for the byte# given (in the BBYTE table). The number of successive entries filled in BBYTE is given by count.

Process 1553 command list

The pointer is used to process a sequence of 1553 command blocks, each of which executes one 1553 command. Each command may result in up to 32 data words transferred. The *count* word in this case indicates the number of command blocks to be processed. For each word read by a command block, a new reading is stored in consecutive channels. In the binary data case, with the table#=5, each word read produces two consecutive bytes of binary status readings. Separate queues are maintained of commands awaiting execution by multiple 1553 controllers. The interrupt following completion of one command passes the next command, if any, to the controller.

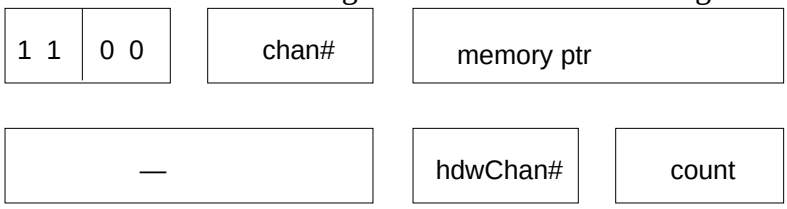
Wait, post-process 1553 data



Wait for the 1553 interrupt activity to complete. Systems which do 1553 I/O with interrupts allow overlapping of multiple 1553 controller activity during DAT processing. This entry must be used to wait for all the readings which have been queued up for interrupt-driven acquisition to finish. This post-processing of 1553 data collection also copies the data into the readings field of the ADATA table, so that this entry must be included. The *timeout* word specifies the time within the cycle (in 0.5 ms units) after which to give up awaiting all controllers in the range specified by *ctrlr#* and *count* and continue DAT processing of any remaining entries.

Analog Devices A/D board

This is the driver for a VME digitizer board from Analog Devices.



Here, *memory ptr* is the base address of the board. and *hdwChan#* is the initial hardware channel select.

Send data request to SRMs

2 0	0 0	—	—
SRMnode#	reqType	#bytes	—

Smart Rack Monitors (SRMs) are used in the Fermilab Linac. As many as 5 SRMs are connected via ARCnet to a single VME station. This entry sends out a request message for data to be returned from an SRM. The *SRMnode#* is usually $7A00, specifying broadcast to all SRMs. The *reqType* is $2201 in the case of requesting the SRM to read and return all its normal cycle data. The *#bytes* specifies the maximum size of the return data buffer. This DAT entry does not wait for the response from the SRMs. That function is specified using the next entry. For more details, see the document SRM Message Protocols.

Wait for SRM data reply

2 1	0 0	—	—
SRMnode#	deadLine	—	—

Await responses from a specific SRM. The *deadline* word specifies the maximum time within the current cycle to wait, in 0.5 ms units. In response to a broadcast request, the order of SRM responses is not determined. But the system's SRM support knows which have responded since the request was sent. It is necessary to place this DAT entry before any $22 entries that refer to the same SRM node#. See the document SRM Message Protocols for more details.

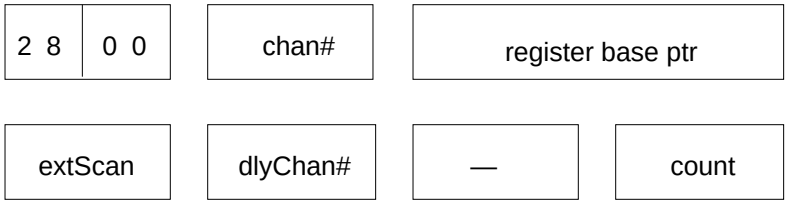
Map SRM data into data pool

2 2	0 0	chan#	—
SRMnode#	offset	table#	offset	count
2 2	0 5	byte#	—
SRMnode#	offset	table#	offset	count

These entries process the already-received response data from an SRM and copy it

selectively into the data pool. The *table#/offset* word identifies the SRM data segment of the response buffer that is to be mapped to the channel or byte data. See the document SRM Message Protocols for more details.

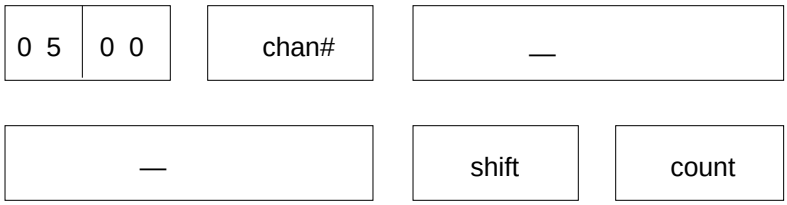
Copy from IRM A/D circular buffer



The IRM analog IndustryPack board maintains a 64K-byte memory that is updated by the hardware with 64 channels of analog input digitized and stored every millisecond. There is room for 512 samples of such data, covering about 0.5 second of time. This entry usually copies the most recently-digitized set of 64 readings into the data pool. If *dlyChan#* is nonzero, it backs up to a time within the current cycle given by the reading of the indicated channel. If *extScan* has the least bit set, it causes the A/D interface to use an external trigger for its digitaizer scan. This option is needed for the PET project, where the scan rate is 360Hz rather than 1000Hz. The *register base ptr* refers to the analog IP board's register address. It is usually FFF58300.

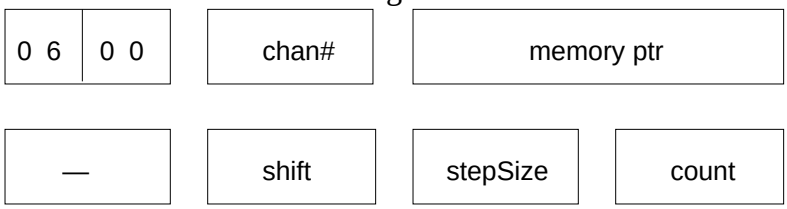
Modify/compute data already acquired

Shift data words



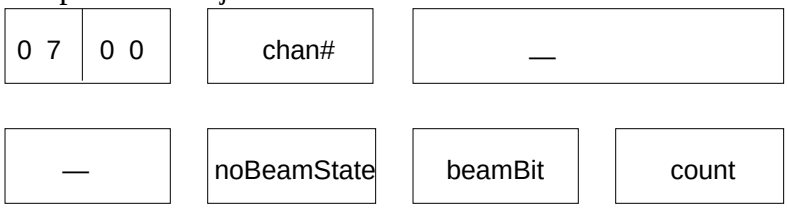
Shift reading fields of a sequence of channels. If *shift* is negative, right shift reading word with sign extension. If *shift* is positive, left shift reading word with zero fill. This has been used to adjust 12-bit A/D readings based on a 2.5 volt scale so that they appear to come from a 14-bit A/D with a 10 volt scale. This is a replace operation.

Adjust nonlinear RF diode readings



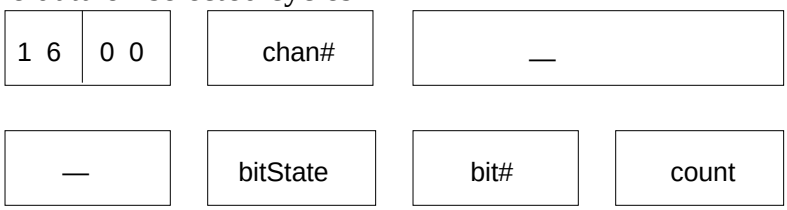
Certain RF amplitude and power readings encountered in the Fermilab Linac system were measured by detector diodes and therefore have nonlinear characteristics. This entry linearizes the readings so they can be linearly scaled in higher level programs the same as any other analog channel readings. Channels in the indicated range from *chan#* to *chan#+count-1* were linearized according to one of two formulae if specified by flags in the "conversion flags" field of the analog descriptor. Flag bit#3 specifies that linearization is to be performed; bit#0 specifies either gradient (0) or power (1) linearization algorithm. If *stepSize* is nonzero, the nonlinear data is taken from memory beginning at memory ptr rather than from the present reading field of the target channel. The *shift* word specifies a shift applied to the raw data word before linearization. See document xxxxx for more details on the linearization algorithms used.

Zero-data pedestal adjustment



Perform automatic pedestal subtraction for selected channels in the target range specified by chan# and count. If *beamBit* is nonzero, it is an optional beam status bit whose no-beam state is given by the sign bit (bit#15) of *noBeamState*.. If *beamBit* is zero, the default beam status Bit# (009F) and no-beam state ($8000) will be used. Each channel to be so treated must be indicated by the appropriate flag bit set (bit#2) in the "conversion flags" byte in the analog descriptor. The result of this logic is that readings read exactly zero, by definition, for cycles in which there is no beam. The pedestal value is kept in the setting word of each channel so treated, so the channel cannot be settable. It may, however, have motor control, since motor-controlled channels do not have setting values. In order for the *beamBit* status to be valid, this entry should occur in the DAT *after* the type 04 entry that updates the BBYTE table with binary status bytes.

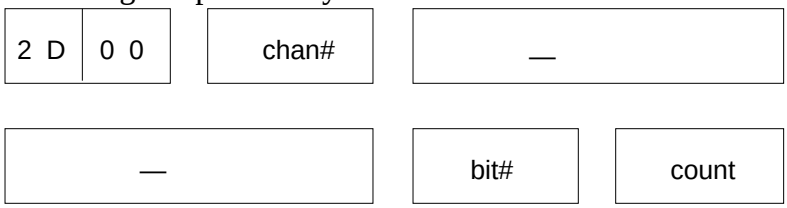
Capture data on selected cycles



Scan the readings of a sequence of channels and capture the reading values for each channel in the range that is marked to need this treatment in its analog descriptor via bit#1 of the "conversion flags" byte. The capture is done on cycles when the status *bit#* matches the bit state given (in the sign bit of *bitState*); otherwise, the captured reading is copied over the current reading, thus

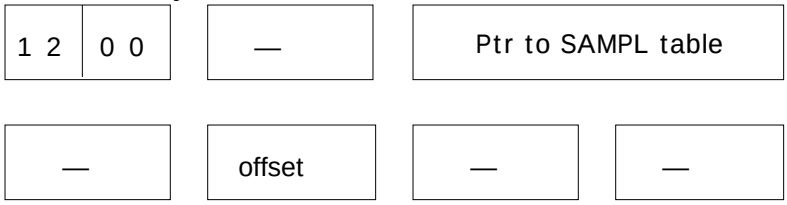
preserving the reading that had been captured before. In systems with channels whose data is valid only during some selected cycles, this entry allows preserving only valid data readings in the local data pool. When a host computer requests data from such channels, it will find only the most recent valid readings there. If it is necessary to also have the current readings, another channel with a copy of the same channel's reading could be set up normally. The captured data values are written into the 7th word of an ADATA entry. Note that motor control cannot be used for such channels, since that same word is used as a motor countdown word in that case.

Save all readings in present cycle



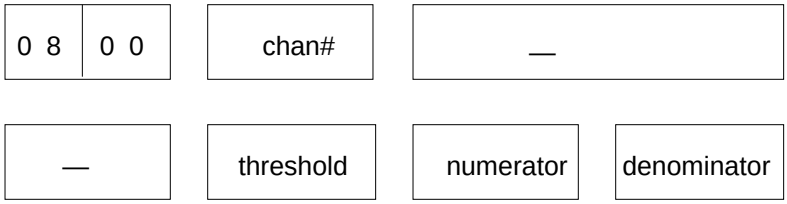
For the range of selected channels specified by *chan#* and *count*, capture the present reading fields into the 8th word of the ADATA entries. The *bit#* word specifies in the lo 15 bits the status bit# that determines whether this capture operation is performed. The state is indicated in the ms bit of this same word. After capture, a host may want to retrieve these values using the listype# defined for accessing the 8th word of an ADATA entry—before the next occurrence of the same status bit state.

Sample data facility



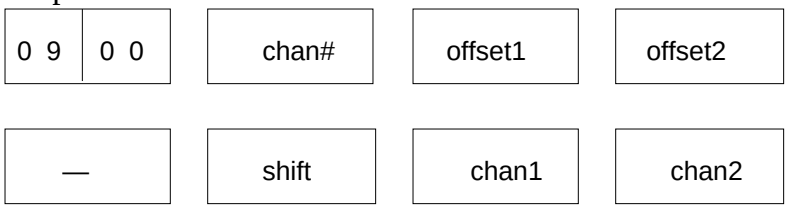
From parameters stored in the SAMPL table, copy a set of channel readings from the local station to memory (especially on the Vertical Interconnect). A table is built containing pairs of words, each of which has a channel# word followed by the data value word. Additional details on this are found in the document Sample Data Facility for VME Stations. This was used by D0 in the "early days."

Compute ratio



Compute ratio between two analog channel readings, where *numerator* and *denominator* are channel#s, and *theshold* is the value of the denominator channel such that, if the absolute value of the denominator reading is below it, the result *chan#* reading will be zero; otherwise the result will be numerator / denominator expressed in volts; i.e., if the readings are equal, the result will be one volt, or $0CCC. The standard full scale range is 10 volts. If an overflow results, use +/– full scale, as appropriate.

Compute product

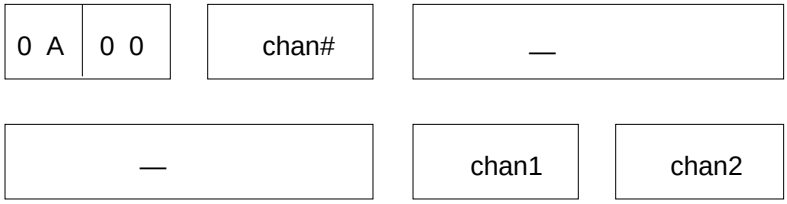


Compute product of two channels *chan1* and *chan2*, and scale by shift. The complete formula used is:

(*chan1*.reading–*offset1*)*(*chan2*.reading–*offset2*)*2^*shift*

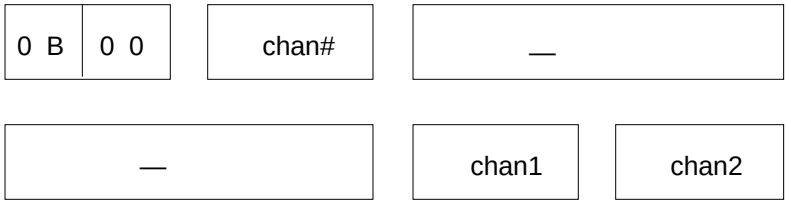
Note that the values of the two offsets are constants, not channel#s.

Compute sum



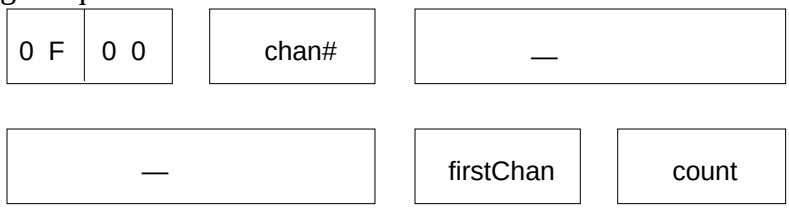
Compute sum of two channels *chan1* + *chan2*. Divide result by 2 in order to prevent overflow. As an example, if the full scales of two readings were both 100.0 amps, then to derive a *chan#* reading that is the sum of the two, the full scale of the result channel should be 200.0 amps.

Compute difference



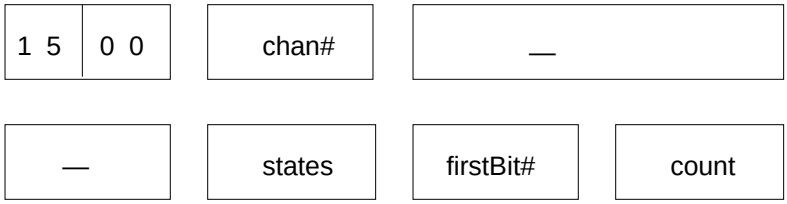
Compute difference of two channels *chan1 – chan2*. Divide result by 2 in order to prevent overflow. As an example, if the full scales of two readings were both 100.0 amps, then to derive a *chan#* reading that is the difference of the two, the full scale of the result channel should be 200.0 amps.

Average sequence of channels



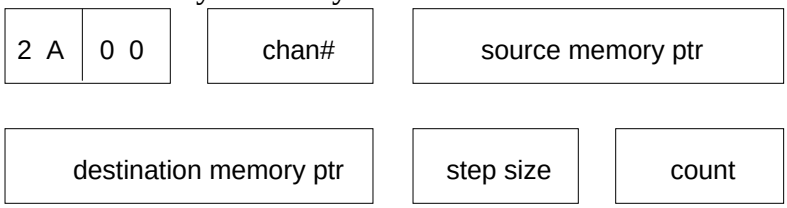
Average the sequence of channel readings from *firstChan* to *firstChan+count–1* and place the result in the reading field of*chan#*.

Beam status counter



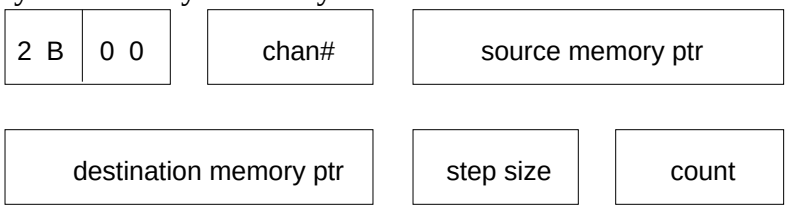
Produce counter readings in target channels by sampling bit# states of sequential status bit#s. The first bit state, to be compared with the *firstBit#* status, is in the sign bit of the states word. Successive status bit#s are compared to successively lower-numbered bits in the *states* word. This naturally limits *count* to 16. When a status bit matches the indicated state, the counter is cleared; when it differs, the counter is incremented. One use of this would be to build a channel whose reading is a counter that measures the #cycles since the last beam cycle. This feature is described further in the document Monitoring Counters.

Copy words memory–memory



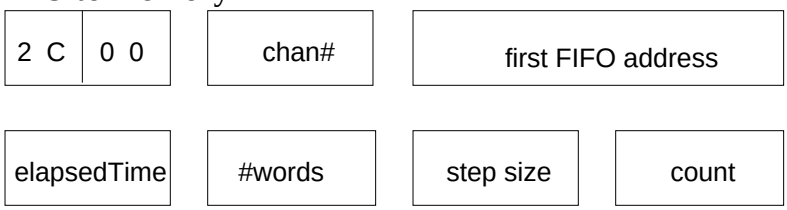
Copy *count* memory words from *source memory ptr* to *destination memory ptr*. The *step size* is used to advance the *source memory ptr*. If the source words are consecutive, then *step size* = 2.

Copy bytes memory–memory



Copy *count* memory bytes from *source memory ptr* to *destination memory ptr*. The *step size* is used to advance the *source memory ptr*. If the source bytes are consecutive, then *step size* = 1.

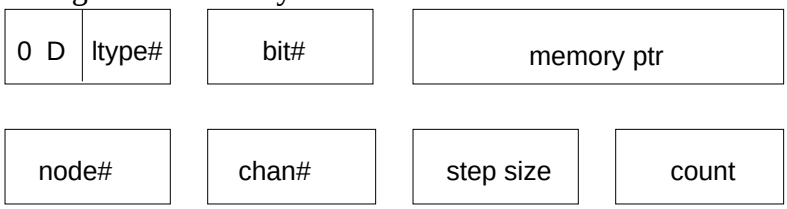
Copy FIFO to memory



Copy FIFOs contents into memory. The *count* word specifies how many FIFOs to read out. The *#words* specifies how many words to read out from each FIFO. The *step size* is used to advance to th next FIFO address. The destination address is found from the analog control field of each *chan#* in sequence. Such channels may be called waveform channels. (This method was used for the first version of the swift digitizer IP module. A later version added memory to the board so that readout of the FIFOs by the *CPU* was no longer necessary.)

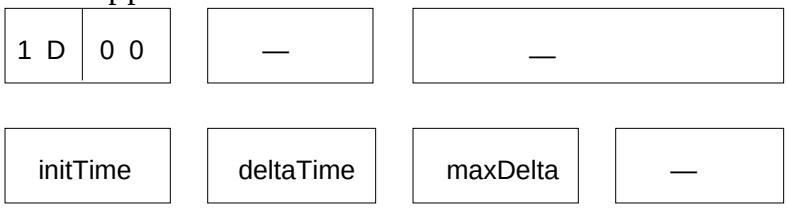
Miscellaneous

Auto-setting from memory



Memory words are copied as setting data, where the channels to be set are consecutive starting at *chan#* in *node#*. This is a way to turn memory data words into readings in another station, although it can, of course, also reference channels in the local station by using the local node#. If the *bit#* (with state value in the sign bit) is nonzero, it conditions the setting action upon the state of the indicated status bit.

Invoke local applications



Scan all entries of LATBL (local application table) in sequence. For each entry that is enabled, call the named local application, including the appropriate value for the call type: initialize, terminate, or cycle. In this way, every enabled local application is invoked every cycle, giving it a chance to perform whatever it needs to do on that cycle. Every LA instance must specify an enable Bit# as the first parameter in its LATBL entry. When the bit is set, the instance is enabled. The three time word indicated above are diagnostics, all in 0.5 ms units. The *initTime* word is the time of starting this DAT entry within the current cycle. The *deltaTime* word is the total cpu time used by all the enabled LA's this cycle. The *maxDelta* word is the maximum value of *deltaTime* ever. Since this DAT entry executes closed loops, it is usually one of the last entries in the DAT, so that it has access to the latest values in the data pool.